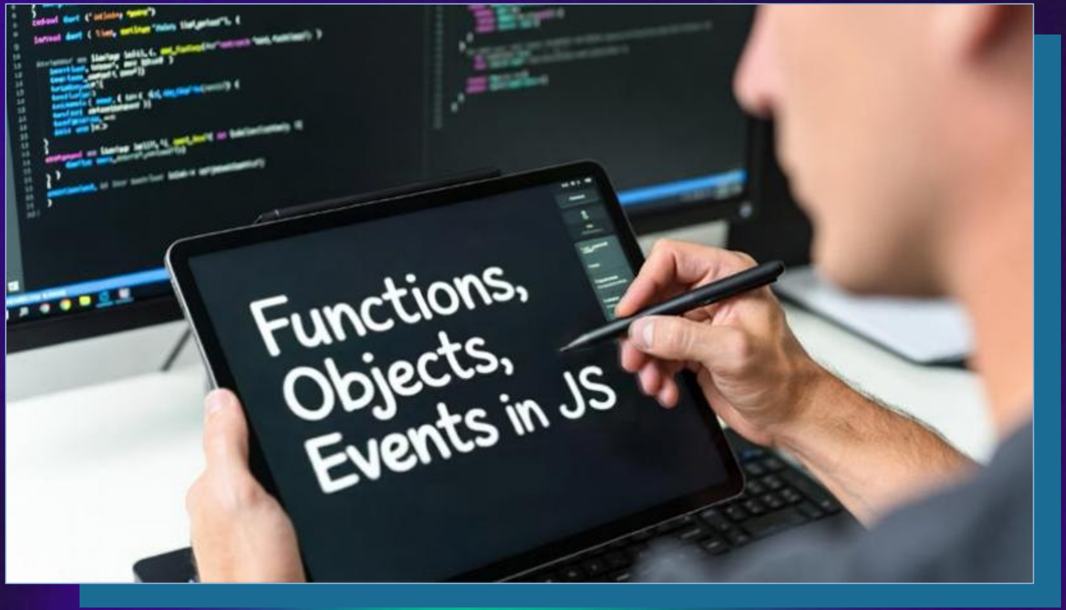
Functions, Objects, Events

Functions, Arrow Functions, Callbacks, Objects, Classes, Constructors, Methods, Events



Svetlin Nakov, PhD

Co-founder @ SoftUni

SoftUni AI

https://ai.softuni.bg

Agenda

1. **Functions** in JS
   - Functions, Parameters, Return Value, Defining and Invoking
   - Arrow Functions, Functions as Parameters, Callbacks
2. **Objects** in JS
   - Keeping Related Values Together
3. Intro to **Classes** in JS: Defining and Using Classes
   - Classes, Fields, Constructors, Methods
4. **Events** and **Event Handling**
   - Event Handling in HTML and JavaScript

Sli.do Code

#AI-Programming

Join at

slido.com

#AI-Programming

Breaks

20:00 / 21:00

Functions in JS

Defining and Invoking Functions, Recursive Functions, Debugging

Functions in JavaScript

- In JavaScript, a **function** is **named block of code**

```
function printHello() {
  console.log("Hello!");
  console.log("I am a function.");
}
```

- Once defined, a function can be **invoked** multiple times

```
for (let i = 1; i < 10; i++) {
  printHello();
}
```

Functions in JavaScript

- Functions can take **parameters** and **return** values

```
function calcCircleArea(radius) {
  let area = Math.PI * radius * radius;
  return area;
}
```

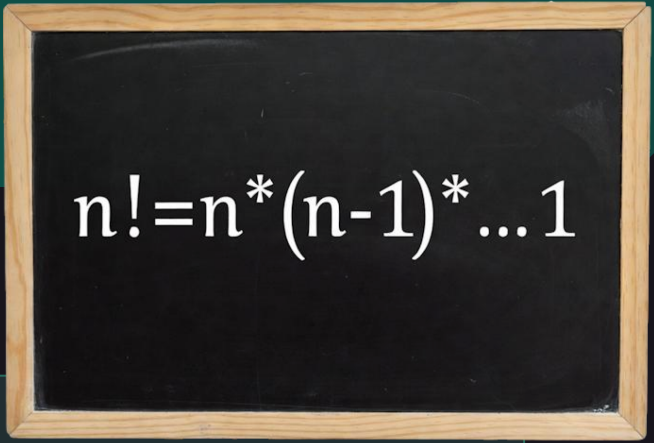
- Once defined, a function can be **invoked** multiple times with different parameters

```
console.log(calcCircleArea(5));
console.log(calcCircleArea(12.5));
```

Recursive Functions

- A function can **invoke itself** recursively:

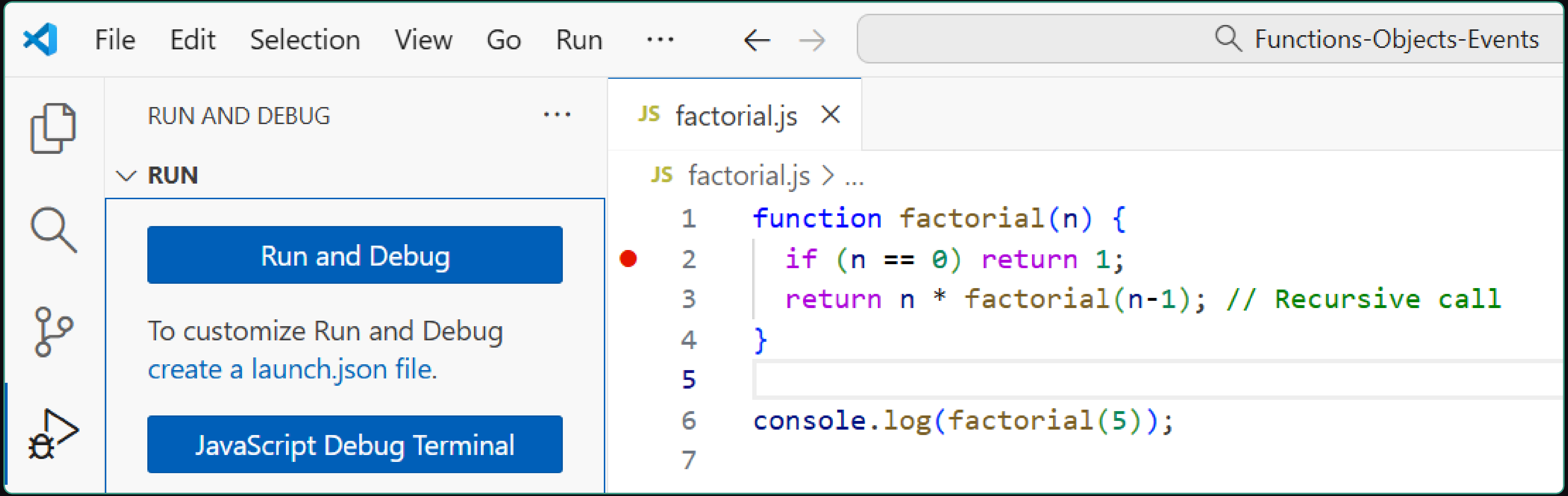
```
function factorial(n) {
  if (n == 0) return 1; // Base case
  return n * factorial(n-1); // Recursive call
}

console.log(factorial(5)); // 120
```



- **Recursion** is a powerful technique in programming
- Recursive functions should have a **base case** to avoid **infinite recursion**

Debugging & Breakpoints in VS Code

- **Debugging** == tracing the code execution to find bugs



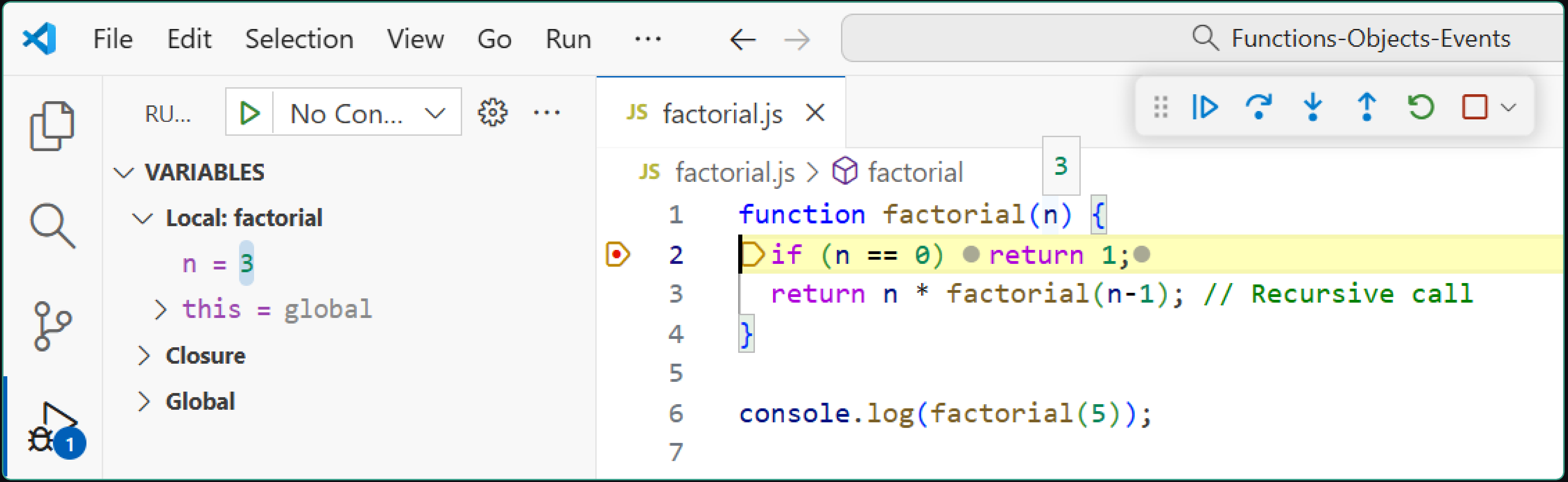
To customize Run and Debug create a launch.json file.

```javascript
function factorial(n) {
  if (n == 0) return 1;
  return n * factorial(n-1); // Recursive call
}

console.log(factorial(5));
```

- **Breakpoint** == intentional **pause-point** in the code
  - Stops the code execution to inspect the internal state

Using the VS Code Debugger

- Inspecting the internal **execution state**



VARIABLES
Local: factorial
n = 3
this = global
Closure
Global

```js
function factorial(n) {
  if (n == 0) return 1;
  return n * factorial(n-1); // Recursive call
}

console.log(factorial(5));
```

- **Watch** / modify variables, view the **call stack**
- **[F5]** → Continue, **[F10]** → Step Over, **[F11]** → Step Into

Problem: Big Factorial

- Write a JS function for **calculate n!** (factorial)
  - Ensure it works for **big inputs** (e. g. 50 factorial)
- **Solution**: we shall use **BigInt** arithmetic

```
function factorial(n) {
  if (n == 0 || n == 1) return 1n;
  return BigInt(n) * factorial(n - 1);
}
console.log(String(factorial(50)));
// 30414093201713378043612608166064768844377641568960512000000000000
```

n!

Judge link: https://alpha.judge.softuni.org/contests/functions-objects-events/5273

Variable Number of Arguments

```
function sum(...numbers) {
  let total = 0;
  for (let num of numbers)
    total += num;
  return total;
}

console.log(sum(2, 3)); // 5
console.log(sum(2, 3, 4, 5)); // 14
console.log(sum(3)); // 3
console.log(sum()); // 0
```

Variable number of arguments

for-of loop enumerates the input arguments

Problem: Incomes and Expenses

- We are given a sequence of **commands**:
  - **Income: {sum}**
  - **Spend: {sum}**
- Write a function to **process all commands** and calculate the final balance (starting with 0 initially)

```
console.log(processExpenses(
   "Income: 50", "Income: 70", "Expense: 30",
   "Income: 100", "Expense: 40", "Income: 50");
// 200
```

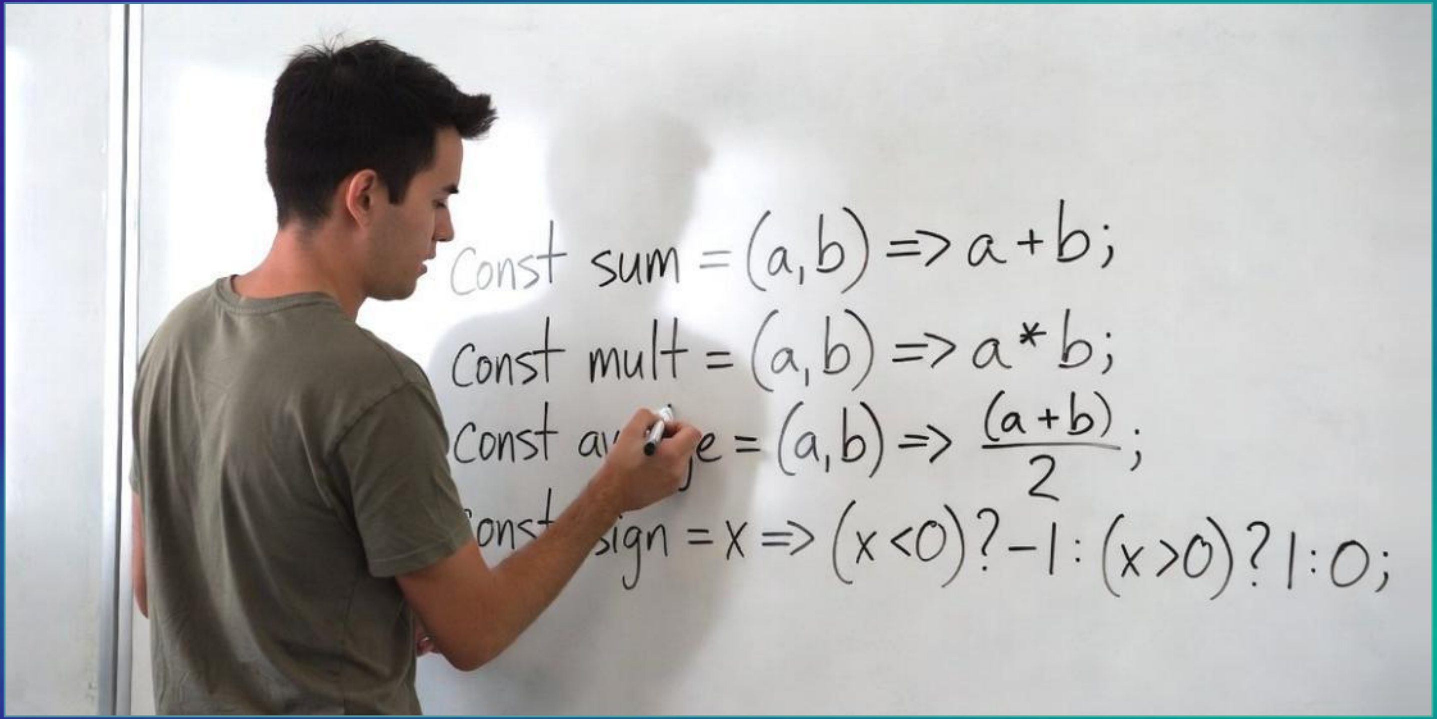
Solution: Incomes and Expenses

```
function processExpenses(...commands) {
  let balance = 0;
  for (let command of commands) {
    const [type, amount] = command.split(": ");
    const value = parseFloat(amount);
    if (type == "Income")
      balance += value;
    else if (type == "Expense")
      balance -= value;
  }
  return balance;
}
```

Judge link: https://alpha.judge.softuni.org/contests/functions-objects-events/5273

Arrow Functions

Function Expressions, Arrow Functions, Higher-Order Functions



```
Const sum = (a,b) => a+b;
Const mult = (a,b) => a*b;
Const av  ge = (a,b) => (a+b) ;
                         2
 onst  sign = x => (x<0)?-1:(x>0)?1:0;
```

Function Expressions

`f = function(…) {…}`

SoftUni AI

- Variables can hold **values** of type "**function**"
- Defined through a **function expression**

```
const add = function(a, b) {
  return a + b;
}

console.log(add); // [Function: add]
console.log(add(2, 3)); // 5

let sum = add, sqrt = Math.sqrt;
console.log(sqrt(sum(8, 1))); // 3
```

Arrow Functions

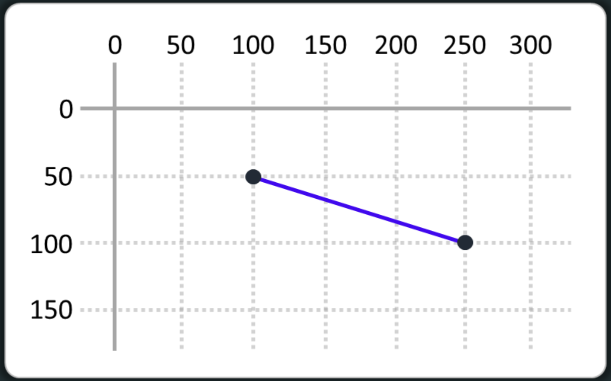
`(x) => 2 * x`

- **Arrow functions** (lambda) use the arrow operator `=>` to provide a **shorter syntax** for function expressions:

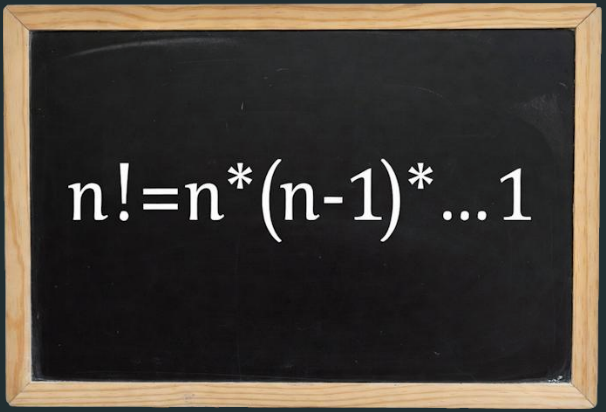
```
const sum = (a, b) => a + b;
const mult = (a, b) => a * b;
const average = (a, b) => (a + b) / 2;
const sign = x => (x < 0) ? -1 : (x > 0) ? 1 : 0;

console.log(sum(2, 5), mult(2, 5)); // 7 10
console.log(average(sum(2, 5), mult(2, 5))); // 8.5
console.log(sign(3), sign(0), sign(-4)); // 1 0 -1
```

Arrow Functions – More Examples

```
const distance = (x1, y1, x2, y2) =>
  Math.sqrt((x2-x1)**2 + (y2-y1)**2);

console.log(distance(100, 50, 250, 100));
// 158.11388300841898
```



```
const factorial = (n) => {
  if (n == 0) return 1;
  return n * factorial(x-1);
}

console.log(factorial(5)); // 120
```



Anonymous Functions

SoftUni AI

- **Anonymous functions** have no name:

```
console.log(function(x) { return x * x });
// [Function (anonymous)] (unnamed function)
```

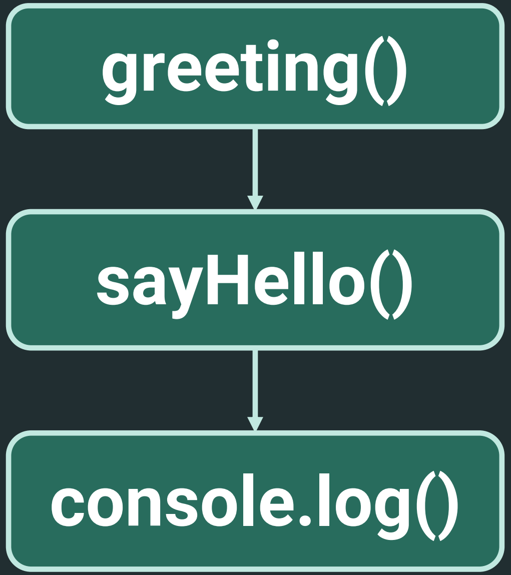
```
console.log(() => console.log("hello"));
// [Function (anonymous)] (unnamed function)
```

```
let sum = (a, b, c) => a + b + c;
console.log(sum);
// [Function: sum] (has name "sum", not anonymous)
```

Functions as Arguments

- **Functions** can be passed as **arguments** to other functions:

```
const sayHello = () => "Hello, ";
const sayHi = () => "Hi, ";
const sayBye = () => "Bye, ";

function greeting(greetingFunc, name) {
  return greetingFunc() + name;
}

console.log(greeting(sayHello, "JS!")); // Hello, JS!
console.log(greeting(sayHi, "JS!")); // Hi, JS!
console.log(greeting(sayBye, "JS!")); // Bye, JS!
```



Higher-Order Functions

f(g(x))

SoftUni AI

- **Higher-order functions** work by invoking other functions, passed as arguments (or return a function as a result)

```
function aggregate(start, end, operation) {
  for (var result = start, i = start+1; i <= end; i++)
    result = operation(result, i);
  return result;
}
```

```
console.log(aggregate(1, 5, (a, b) => a + b)); // 55
console.log(aggregate(1, 5, (a, b) => a * b)); // 120
console.log(aggregate(1, 5, (a, b) => '' + a + b)); // 12345
```

Problem: Special Numbers

- Write a function to return all numbers in range [start … end]
  - Divisible to **3**
  - Containing digit **2**
- Use **higher-order function**: iterate over the range in a loop and filter the range with arrow function

```
console.log(specialNumbers(20, 30));
// Nums: 21 24 27
```

```
console.log(specialNumbers(100, 200));
// Nums: 102 120 123 126 129 132 162 192
```

Solution: Special Numbers

```
function specialNumbers(start, end) {

  function generateRange(start, end, filter) {
    let result = '';
    for (let num = start; num <= end; num++)
      if (filter(num))
        result += (result ? ' ' : '') + num;
    return result;
  }

  let filterDiv3 = (num) => num % 3 == 0;
  let filterContains2 = (num) => num.toString().includes('2');
  let filters = (num) => filterDiv3(num) && filterContains2(num);

  return "Nums: " + generateRange(start, end, filters);
}
```

Judge link: https://alpha.judge.softuni.org/contests/functions-objects-events/5273

Function Returned by Function

- A **function can return another function** as output:

```
function greeting(message) {
  return function(name) {
    return message + name;
  }
}

let sayHi = greeting("Hi, ");
console.log(sayHi); // [Function: anonymous]
console.log(sayHi("Steve")); // Hi, Steve

let sayWelcome = greeting("Welcome, ");
console.log(sayWelcome("Steve")); // Welcome, Steve
```

```
f(x) => function g(x)
```

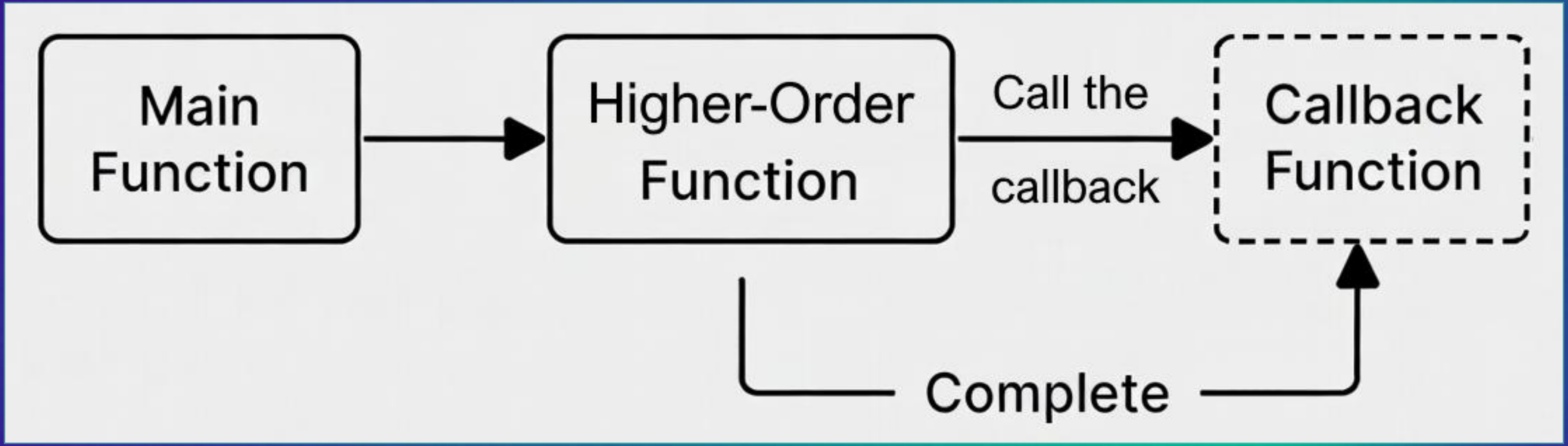
Closures

- **Closure** == function returning a function, with an **internal state**

```
function createCounter(start) {
  let count = start; // Internal state: count
  return function() {
    return count++;
  }
}
let counter1 = createCounter(100);
console.log(counter1(), counter1()); // 100 101
let counter2 = createCounter(1);
console.log(counter2(), counter2()); // 1 2
```

Callback Functions

Functions, Sent as Arguments,
Designed to be "Called Back"



Callbacks

- In programming, a **callback function** is a function, given as parameter, indented to be "***called back***"

```
function scanRange(start, end,
    onStart, onNumber, onEnd) {
  onStart(); // Invoke a callback
  for (let i = start; i <= end; i++) {
    onNumber(i); // Invoke a callback
  }
  onEnd(); // Invoke a callback
}
```

Using Callback Functions

SoftUni AI

- Invoking a function, which requires **callback functions**:

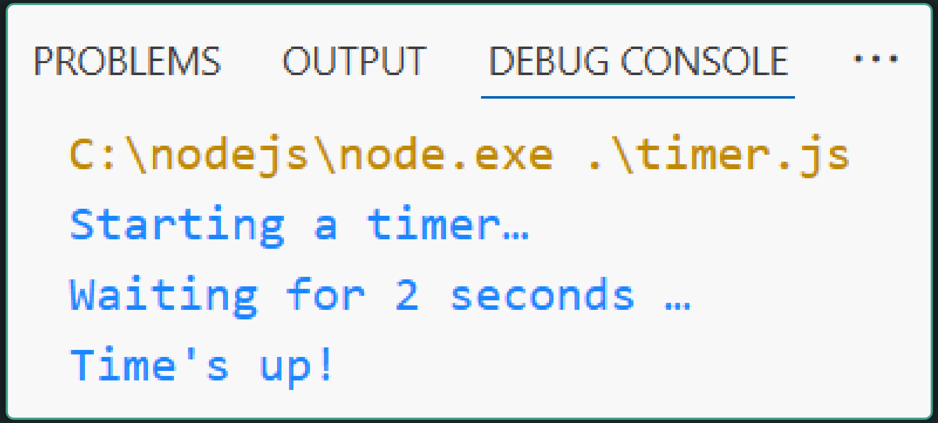
```
scanRange(1, 3,
  () => console.log("Starting scan..."),
  (num) => console.log("Found number: " + num),
  () => console.log("Scan complete.")
);
```

```
Starting scan...
Found number: 1
Found number: 2
Found number: 3
Scan complete.
```

Built-In Callbacks in JavaScript

- In JavaScript **callbacks** are highly popular, for example:
  - **setTimeout(…)** will invoke a **callback** after time elapsed

```
console.log("Starting a timer…");

function timeoutCallback() {
  console.log("Time's up!");
}

setTimeout(timeoutCallback, 2000);

console.log("Waiting for 2 seconds …");
```



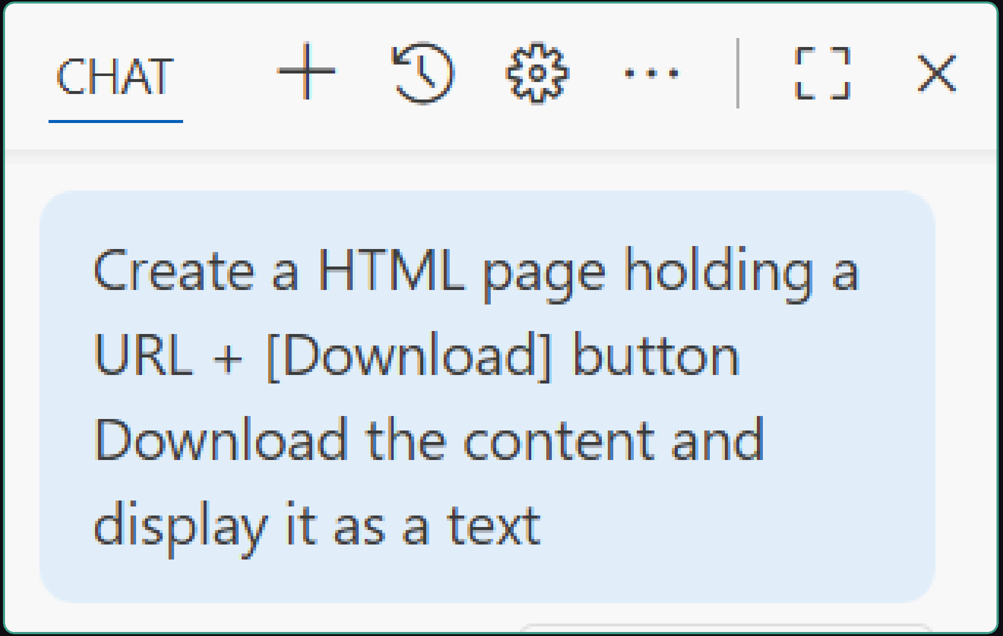
```
C:\nodejs\node.exe .\timer.js
Starting a timer…
Waiting for 2 seconds …
Time's up!
```

Real-World Callbacks in JS

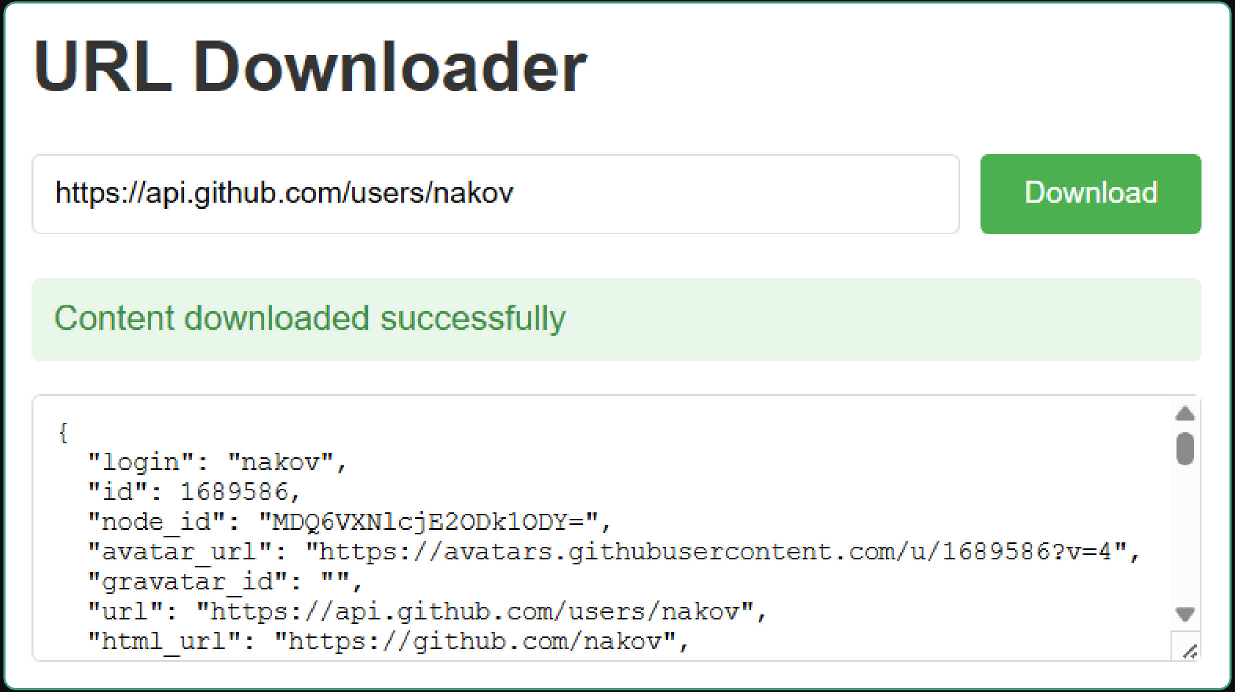
```
const https = require('https');
let request = https.get('https://softuni.org');
request.on('response', function(response) {
  let data = '';
  response.on('data', chunk => data += chunk);
  response.on('end', () => console.log(data));
});
request.on('error', function(error) {
  console.log('Network error: ' + error.message);
});
```

Problem: URL Downloader

- Create a **HTML page** holding a **URL** + **[Download]** button
  - Download the content and display it as a text
- **Solution**: prompt **Copilot**



Create a HTML page holding a URL + [Download] button Download the content and display it as a text



URL Downloader

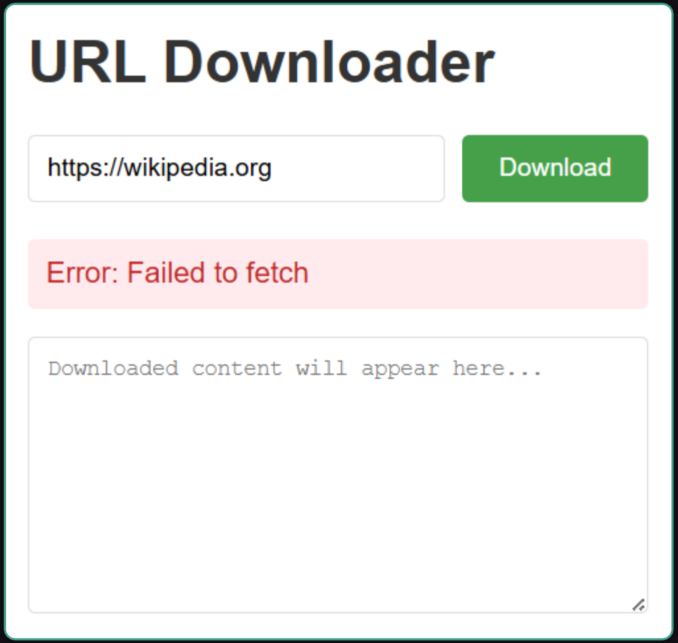
https://api.github.com/users/nakov

Content downloaded successfully

```
{
  "login": "nakov",
  "id": 1689586,
  "node_id": "MDQ6VXNlcjE2ODk1ODY=",
  "avatar_url": "https://avatars.githubusercontent.com/u/1689586?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/nakov",
  "html_url": "https://github.com/nakov",
```

URL Downloader Fails

- The "**URL Downloader**" app **fails** to load most Web sites:
  - Loading `**https://wikipedia.org**` → Error: Failed to fetch
- This is a **security limitation** in Web browsers: **CORS policy**


URL Downloader

https://wikipedia.org

Error: Failed to fetch

```
Downloaded content will appear here...
```

- Use CORS-enabled URL addresses, e.g.
  - https://api.github.com/users/nakov
  - https://catfact.ninja/fact
  - https://api.chucknorris.io/jokes/random
  - https://pokeapi.co/api/v2/pokemon/pikachu

Objects in JavaScript

Keeping a Set of Fields Together

person	
name	Maria
age	24
town	Sofia

```
let person = {
  name: "Maria",
  age: 24,
  town: "Sofia"
};
```

Objects in JavaScript

- **Objects** in JavaScript keep a set of values together

```
let person = {
   name: "Maria",
   age: 24,
   town: "Sofia"
};
```

person	
name	Maria
age	24
town	Sofia

```
person.age++;
console.log(`I am ${person.name}.`);
console.log(`I am ${person.age} years old.`);
```

Working with Objects

```
let person = { name: "Maria", age: 24 };
person.isStudent = true; // Add a new property
console.log(person);
// { name: 'Maria', age: 24, isStudent: true }
person.age++; // Change a property
console.log(person['age']); // Access by index → 25
delete person.isStudent; // Remove a property
console.log(person); // { name: 'Maria', age: 25 }
for (let key in person) // Loop through the object
  console.log(key + ": " + person[key]);
```

Nested Objects

Nested object: object inside another object

```
const user = {
  name: "George",
  address: { town: "Plovdiv", country: "BG" },
};
console.log(user);

user.username = "gogo98";
let job = { title: "Developer", company: "SoftUni" }
user.job = job;
user.job.title = "CTO";
console.log(user);
```

JSON

- **JSON** is a text-format for storing JavaScript objects

```
const user = {
  name: "George",
  address: { town: "Plovdiv", country: "BG" },
};
const userJSON = JSON.stringify(user);
console.log(userJSON); // userJSON is string
const colorJSON = '{"red":75,"green":89,"blue":43}';
let c = JSON.parse(colorJSON);
console.log(`RGB(${c.red}, ${c.green}, ${c.blue})`);
```

Problem: Largest Rectangle

- Write a function to take **several rectangles**, given as **JSON** and print the **largest** of them

```
printLargestRectangle(
  '{"width":30, "height":20}',
  '{"width":5, "height":120}',
  '{"width":15, "height":40}',
  '{"width":25, "height":25}',
  '{"width":35, "height":15}',
)
```

```
Largest rectangle: 25 x 25 -> area: 625
```

Solution: Largest Rectangle

```
function printLargerRectangle(...rectanglesAsJSON) {
  let largestArea = 0, largestRect = null;
  for (let rectJson of rectanglesAsJSON) {
    const rect = JSON.parse(rectJson);
    const area = rect.width * rect.height;
    if (area > largestArea)
      [largestArea, largestRectangle ] = [area, rect];
  }
  if (largestRectangle)
    console.log(`Largest rectangle: ${largestRect.width} x
      ${largestRect.height} -> area: ${largestArea}`);
}
```

Judge link: https://alpha.judge.softuni.org/contests/functions-objects-events/5273

Methods: Functions inside Objects

```
const rectangle = {
  x: 150, y: 40,
  width: 20, height: 15,
  move: function(dx, dy) {
    this.x += dx;
    this.y += dy;
  }
  calcArea() { return this.width * this.height; },
  toString() { return `Rect(${this.x}, ${this.y})`; }
}
```

Method move(dx, dy)

`this` means the object itself

Method toString()

Calling Methods of an Object

```
console.log("" + rectangle); // Invokes toString()
// Rect(150, 40)

rectangle.move(50, -10);
console.log(rectangle.toString()); // Rect(200, 30)

console.log("Area:", rectangle.calcArea()); // Area: 300

for (let i=10; i<=100; i+=10) {
  rectangle.move(i, 0);
  console.log("Rectangle moved to: " + rectangle);
}
```

Objects as Function Parameters

```
function printUser({name, address: {town, country}}) {
  console.log(`Name: ${name}`);
  console.log(`City: ${town}`);
  console.log(`Country: ${country}`);
}

const user = {
  name: "Nina",
  address: { town: "Plovdiv", country: "BG" }
};
printUser(user);
```

This syntax is called "object destructuring"

Objects as Function Result

SoftUni AI

```
function calcStats(...numbers) {
  let [min, max, sum] =
    [Number.POSITIVE_INFINITY, Number.NEGATIVE_INFINITY, 0];
  for (num of numbers) {
    if (num < min) min = num;
    if (num > max) max = num;
    sum += num;
  }
  return { min, max, average: sum / numbers.length, sum };
}
console.log(calcStats(5, 10, 15, 20, 25));
// { min: 5, max: 25, average: 15, sum: 75 }
```

Group declaration + assignment

Return an object

Continued in part 2

Questions?

Postbank – Exclusive Partner for SoftUni AI

- One of the leading **banking institutions** in Bulgaria
- Member of the Eurobank Group with € 99.6 billion of assets
- Innovative trendsetter in the digitalization and transformation
- Certified Top Employer 2024 by the international Top Employers Institute
- AI integration for business, learning and development
- AI Assistants for talents: Story Builder, CV Assistant, Interview Trainer
- Proven people care and wellbeing initiatives
- Benefits and unlimited access to professional, personal and leadership trainings and programs
- www.postbank.bg / careers.postbank.bg

Diamond Partners of Software University

Software University

Coca-Cola HBC Bulgaria

Postbank

top EMPLOYER България Bulgaria 2025 FOR A BETTER WORLD OF WORK

SUPER HOSTING .BG

encorp.ai

createX

DRAFT KINGS THE CROWN IS YOURS

INDEAVR Serving the high achievers

VIVACOM

Diamond Partners of SoftUni Digital

SoftUni Digital

Coca-Cola HBC Bulgaria

Postbank

top EMPLOYER България Bulgaria 2025 FOR A BETTER WORLD OF WORK

SUPER HOSTING .BG

encorp.ai

DRAFT KINGS THE CROWN IS YOURS

createX

INDEAVR Serving the high achievers

VIVACOM

Diamond Partners of SoftUni Digital

SoftUni Digital

HUMAN

IMPULSE MEDIA®

M

1 FOR FIT

NETPEAK
DIGITAL GROWTH PARTNER

ABC DESIGN &
COMMUNICATION

Zahara
dig.it.all

ETIENYANEV
Break Your *Limits*. Live Your *Brand*.

Aleikov
studio

Diamond Partners of SoftUni Creative

SoftUni Creative

Coca-Cola HBC Bulgaria

Postbank

top EMPLOYER България Bulgaria 2025

FOR A BETTER WORLD OF WORK

SUPER HOSTING .BG

encorp.ai

DRAFT KINGS

THE CROWN IS YOURS

createX

INDEAVR

Serving the high achievers

VIVACOM

Organization Partners of SoftUni Creative

SoftUni Creative

dequitas.

Фигмайстор

Studio π

Think Big • Design Smart

дизайнът на нещата

REDV PRINT STUDIO

THE BUCKS TOWN'S WORK